**15L-4195**

**Aena Maryam**

**Advanced Programming**

**Assignment-2**

**Part 1**

**Differences between Vector and ArrayList:**

|  |  |
| --- | --- |
| **Vector** | **ArrayList** |
| Vector is synchronized which means that only a single thread at a time can access the code | ArrayList is not synchronized which means that multiple threads have access to work on the ArrayList at the same time. |
| Vector operations are slower due to the synchronization ability of vectors. While a thread is already progress on the vector, it acquires a lock on it forcing thee other threads wanting to work on it to wait until the lock is released. | ArrayList is faster as it is non-synchronized. So, threads do not have to wait for the lock to be released. They can carry out their computations simultaneously. |
| Vector increments 100% of the current size if the number of elements exceed its due capacity. | ArrayList increments 50% of the current array size if the number of elements exceed its due capacity. |
| For traversing over the elements of vector, both Enumeration and Iterator can be used. | ArrayList can only use Iterator for traversing. |

**Part 2**

**Differences between HashSet and SortedSet:**

|  |  |
| --- | --- |
| **HashSet** | **SortedSet** |
| Hashset’s Time Complexity is O(1). It uses a hash table and does basic operations faster than the SortedSet. | Sorted Set’s Time Complexity is O(log n) on average case. It uses a binary search tree or a red back tree which is a balanced binary tree and has a far worse performance than HashSet. |
| Continuing the point 1, HashSet does basic operations independent of the size of input data in a constant period of time. | Continuing the point 1, SortedSet does its basic operations logarithmically, while being dependent on the size of input. |
| HashSet is used when we do not want our elements to be sorted. | If we want our elements to be sorted then using SortedSet is preferable. |

**Part 3**

**Differences between HashSet and TreeSet:**

|  |  |
| --- | --- |
| **HashSet** | **TreeSet** |
| HashSet is faster than TreeSet as it takes constant time for operations like search, insert and delete. | TreeSet is slower as it takes O(Log n) time for search, insert and delete operations. It also uses operations like higher(), floor(), ceiling() (which also takes O(Log n) time) etc which are not supported in HashSet. |
| It is implemented using a hash table. | It is implemented using a Self Balancing Binary Search Tree (Red-Black Tree). |
| HashSet Elements are not ordered. | TreeSet maintains its objects/elements are always sorted in ascending order by default, defined by either Comparable or Comparator is Java. |
| HashSet easily allows null object. | TreeSet never allow null object. Since TreeSet uses compareTo() method so in case of null object, this function throws java.lang.NullPointerException. |
| For comparing two objects in Set and for detecting duplicates, HashSet uses equals() method. | For comparing two objects in Set and for detecting duplicates, TreeSet uses compareTo() method. |

**Part 4**

**Differences between Array and List:**

|  |  |
| --- | --- |
| **Array** | **List** |
| An array is a type of list and is always a list. | A list is an abstract data type that implies an ordered sequence of values and a list is not always an array. |
| For concepts like indexing and bounds, arrays are a good data structure to gain knowledge from. There is a wide variety of problem spaces where arrays are a perfect fit. | Lists, however are not a standard. |
| Arrays are easy to understand. | Lists have more diversity as they can be implemented in several different ways like ArrayList, LinkedList where each sort of list has different performance characteristics. |
| Arrays occupy n\*size(type) byes where n is the length of the array and size(type) is the size in memory required to store the data type desired. Arrays are pretty inexpensive to create, release and use because they are chunks of memory or we can say a consecutive segment of memory. | On the contrary, most list implementations are a combination of nodes where values and pointers keep the nodes connected. So, we do not need a big chunk of memory as nodes can be scattered at separate parts of memory. |

**Part 5**

**Differences between List and Set:**

|  |  |
| --- | --- |
| **List** | **Set** |
| List is an ordered grouping of elements. | Set is an unordered grouping of elements. |
| List is used for the collection of elements with duplicates. | Set is used for the collection of elements without duplicates. |
| New methods are defined inside List interface. | No new methods are defined inside Set interface, so we have to use Collection interface methods only with Set subclasses. |
| Lists are accessible by using indices | To access the data from Set, it is required to use Iterator only and index based retrieval is not possible for it. It is mainly used whenever uniqueness collection is required. |
| Lists allow null values. It displays elements in insertion order. | Sets allow single null value. It does not maintain any order to display elements and only TreeSet will display elements in ascending order. |

**Part 6**

**Differences between Navigable Map and Navigable Set:**

|  |  |
| --- | --- |
| **NavigableMap** | **NavigableSet** |
| It extends the SortedMap interface that represents a sorted map. | It extends the SortedSet interface and NavigableMap interface provides methods for navigating elements over our NavigableSet. |
| It offers methods for obtaining the elements of the map relative to others in sorted collection. E.g. It has ceilingEntry(K obj) method that returns an entry with the lowest key that is greater or equal to the object passed as the argument. Also, the lastEntry() method, that returns an entry with the greatest key. | For example, NavigableSet has pollFirst() method that retrieves and removes the lowest element in the set. |
| The **NavigableMap** represents a Map which is additionally sorted in terms of a client. This data structure provides an ordered Map and associates its elements with certain keys so that these elements could be obtained by that keys. | The **NavigableSet** interface represents a Set that is sorted in terms of a client. A set is an unordered collection of distinct elements. |